Automatic Selection of Problem Solving Libraries based on Competence Matching

Mauro Gaspari[1], Enrico Motta[2], and Dieter Fensel[3]

University of Bologna[1], Dipartimento di Scienze dell'Informazione**,** Via Mura Anteo Zamboni 7, 40127 Bologna Italy. E-mail: gaspari@cs.unibo.it

The Open University[2], Knowledge Media Institute**,** Walton Hall MK7 6AA, Milton Keynes, United Kingdom**.** E-mail: e.motta@open.ac.uk

University of Karlsruhe[3]**,** Institute AIFB, Kaiserstr. 12, D-76128 Karlsruhe, Germany. E-mail: dfe@aifb.uni-karlsruhe.de

Abstract. We describe an automatic approach to the selection of problem solving libraries based on semantic matching. Problem solving libraries are specified exploiting UPML an architectural description language specialized for knowledge based systems. UPML allows a designer to specify the internal structure and the competence of complex problem solving libraries. This semantic information is used to select libraries which are suitable to solve a given problem exploiting competence matching. In this paper we introduce UPML and we present and discuss an approach to a decidable competence matching rule.

1 Introduction

One of the main goals of the knowledge engineering research community is to develop methods and mechanisms which facilitate the reuse and the sharing of the knowledge that has been already formalised and embedded in specific applications (Motta et al. 1999). An essential aspect of this knowledge concerns the problem solving method that have to be adopted to compute the solution in a knowledge based system (KBS). A *problem solving method* (PSM) decompose the reasoning task of a KBS in a number of subtasks and inference actions that are connected by knowledge roles. In this way a PSM describes the complex architecture implementing the reasoning part of a knowledge-based system.

Several problem solving method libraries are now available (Marcus, 1988; Chandrasekaran et al., 1992; Puppe, 1993; Breuker & Van de Velde, 1994; Musen 1998; Motta, 1997) and a number of problem-solving method specification languages have been proposed, ranging from informal notations (e.g. CML) (Schreiber et al., 1994) to formal modelling languages see (Fensel & van Harmelen, 1994; Fensel, 1995) for surveys.

The aim of the IBROW3 project (Benjamins et al., 1998) is to produce a software broker able to configure semi-automatically an intelligent problem solver out of reusable components. In order to identify the components relevant to an application the broker needs to understand their functionality and relate them to a problem specification. IBROW3 supports the development of reusable specifications by means of the UPML language (Fensel et al., 1998): the Unified Problem-solving Method description Language.

UPML provides a general framework for specifying PSM which includes:
- a set of basic components to specify generic ontologies, tasks, PSMs and domain models: an *ontology* defines the terminology used in the specification; a *task* formalises a problem to be solved; and a *domain model* represents the knowledge which characterise a particular application.
- a set of mechanisms to specify the competence of PSMs: these mechanisms express the semantics of PSMs and are essential to understand if they can be used to solve a given problem (task).
- a set of adapters which can be distinguished in bridges and refiners to instantiate generic components in particular application domains.
- a set of architectural constraints which provide a formal framework to verify the correctness and the completeness of UPML specifications.
- and finally a set of informal guidelines and tools to facilitate the specification of tasks and PSMs.

UPML is a complex language which deals with all the aspects needed to support reusability in knowledge-based system. Thus a description of all the features provided by UPML is out of the limited scope of this paper. For this reason we will only introduce the competence matching mechanisms of the language which are used to express the semantics of PSM libraries. The reader interested to a more complete description of UPML can refer to (Fensel et al. 1999a and 1999b).

The current version of the IBROW3 broker supports a semi-automatic approach to competence matching: the broker tries to select a PSM such that the PSM competence matches a given goal and, if it does not succeed, it asks the user. However, if certain conditions hold in the UPML specification, the problem of matching the goal with the PSM competence can be made fully automatic. In this paper we investigate this issue. We try to individuate a subset of UPML where competence matching is decidable. UPML specifications are translated in LOOM (Brill, 1993) an efficient theorem prover based on a description logic and competence matching is performed exploiting concept subsumption[1] in LOOM.

The paper is organised as follows: in Section 2 we present UPML by means of an example: the specification of a generic library to solve a search problem. In Section 3 we present how this specification can be translated into LOOM to perform

[1] Subsumption is an operation which determines if a given concept is more general than another. It is similar to the subclass concept but more general because is based on logic and not on a static subclass hierarchy.

competence matching. Finally we present related work and some concluding remarks.

2. The specification of a search library in UPML

To illustrate UPML we provide a specification of a problem solving method that performs search in a state space. In this section we only illustrate the constructs which are significant to illustrate competence matching, the reader interested to a complete description of UPML car refer to Fensel et al. 1999a, 1999b.

2.1. Ontologies

Following the UPML approach we need first to fix the terminology used in the PSM. Thus we begin by defining an ontology for characterizing search spaces.

ontology state-space-terminology

pragmatics

This ontology introduces the terminology required for modelling *search spaces*. A search space consists of a number of *states*. Moves along a search space are modelled by means of *state transitions*. We also assume that the input to a search engine is given by some sort of task or domain-dependent input specification. This ontology provides the basic categories of search states, including *initial state*, *deadend state* and *solution state*. It also defines the notions of *path* and *solution path*.

signature

elementary sorts

State, Input;

constructed sorts

State_Space: **set_of** State;

predicates

state_transition: State x State;

initial_state: State;

solution_state: State;

deadend_state: State;

path: State x List(State) x State;

solution_path: State x List(State) x State;

variables

$?s_1$, $?s_2$, $?\ s_3$: State;

?l: List(State);

axioms

/* A state transition is a meaningful move in the state space */

$\forall ?s_1, ?s_2$ (state_transition $(?s_1, ?s_2) \rightarrow ?s_1 \neq ?s_2$);

/* A path is a sequence of state transitions */

path $(?s_1, nil, ?s_2) \leftrightarrow$ state_transition $(?s_1, ?s_2)$;

path $(?s_1, [?s_2 \,.\, ?l], ?s_3) \leftrightarrow$ state_transition $(?s_1, ?s_2) \wedge$ path $(?s_2, ?l, ?s_3)$;

/* An initial state is one which is not the result of any state transition */

initial_state$(?s_2) \leftrightarrow \neg\exists\, ?s_1, ?s_2$ state_transition $(?s_1, ?s_2)$;

/* A deadend state is not a solution state */

deadend_state$(?s_1) \leftrightarrow \neg\exists\, ?s_2, ?l$ (solution_state$(?s_2) \wedge$ path $(?s_1, ?l, ?s_2)$);

/* A solution path is one which leads from an initial to a solution state*/

solution_path $(?s_1, ?l, ?s_2) \leftrightarrow$ path $(?s_1, ?l, ?s_2) \wedge$
initial_state $(?s_1) \wedge$
solution_state $(?s_2)$;

end ontology

The specification of an ontology includes a signature which defines the names and the types of the known operators and a set of axioms which express their meaning.

2.2. Problem solving methods

We can now use the ontology given above to define a task-independent characterisation of problem solving as search. This is achieved defining a search PSM.

The definition of a problem solving method includes:
1. a list of ontologies which define the terminology used in the PSM;
2. some auxiliary definitions;
3. the competence definition which includes: input/output parameters, a precondition, a set of subtasks that must be achieved and a post-condition;
4. an operational definition where the subtasks are used.

Note that a task defines a relation between input and output roles, these relations can be used to define assumptions on task behaviour exploiting first order logic. Also note that in the operational specification below the output argument of tasks is omitted.

problem solving method search

pragmatics

This search method provides a generic control regime that navigates a search space until either a solution state is encountered or state selection fails.

ontology

state-space-terminology;

auxiliary terminology

input: Input;

output: State $\cup$ {$\varnothing$}

states, succ_states: State_Space;

competence

roles

/* It is not entirely clear how to characterise generically the input to a search problem solver. We avoid the problem by assuming that some kind of non-empty input specification is provided */

input input;

output output;

preconditions

input $\neq \varnothing$

subtasks

initialise; select_state; derive_successor_states; update_state_space;

postconditions

/* The output is either a solution state or the empty set, if the PSM fails */

output = $\varnothing$ $\vee$ solution_state (output);

operational description

intermediate roles

state, states, succ_states;

control

begin

states:= **one** x. initialise (**input** input)

repeat

state:= **one** x . select _state (**states** states)

```
if state = ∅ ∨ solution_state (state)
    then
    return state
  else
    succ_states:= one x. derive_successor_states (state state);
    states:= one x. update_state_space (input1 states input2 succ_states)
end if
end repeat
end
```

end problem solving method

We assume the standard semantics for the **repeat** and the conditional statements. The **return** operator stops the execution of the PSM and the **one** operator selects one solution of the related subtask.

2.3. Tasks

Given the specification of the main PSM, then we need to specify the subtasks that are listed in the competence definition. UPML provide a special construct to define tasks which we present showing the definition of the subtask **initialise**. The reader interested to a complete definition of this search PSM can refer to Motta et al. (1999).

task initialise

pragmatics

The goal of this task is to return an initial list of states from an input specification;

ontology

state-space-terminology;

auxiliary terminology

constants

input: Input;

states: State_Space;

variables

?s: State;

specification

roles

input input; **output**: states;

preconditions

input $\neq \emptyset$

goal

states $\neq \emptyset \wedge \forall$?s (?s $\in$ states $\rightarrow$ initial_state (?s));

end task

A task definition includes: input and output parameters (roles), some auxiliary definitions, a precondition and a goal. The precondition is a condition on dynamic inputs and the goal represents what the task wants to achieve. In this case it must generate a set of initial states. Not that the concept of initial state is defined in the associated ontology.

2.4 Competence matching

A typical matching scenario starts from the specification of a task to be solved: the competence matching algorithm tries to select an appropriate PSM which is able to solve it. A possible task specification has the form:

task mysolution

pragmatics

The goal of this task is to find a solution in a state space;

ontology

state-space-terminology;

specification

roles

input input; **output**: state;

preconditions

input $\neq \emptyset$

goal

solution_state (output);

end task

The essential points of this task specification are that the input must not be empty (note that this condition can only be checked at runtime) and that the output must be a solution state.

In UPML task goals and PSM post-conditions are usually expressed exploiting many sorted first order logic. Although first order logic is not decidable in general this choice is adequate to the IBROW3 broker which aims to select (semi-automatically) a

PSM that matches a given task. However if we wont to provide a decidable competence matching mechanism we have to restrict to less expressive formalisms. In the following we illustrate a possible restriction of UPML showing how a competence declaration can be translated into LOOM (Brill, 1993).

3. An automatic approach to competence matching

To illustrate our automatic approach to competence matching we exploit the following restricted version of the UPML task and PSM constructs which only deal with the features used for competence matching.

DEFINITION 1: **TASK**
A Task T includes a goal T.GOAL, a set of input role types $T.INPUT_1 \ldots T.INPUT_N$, where T.# = N, and an output role type T.OUTPUT.

DEFINITION 2: **PSM**
A PSM P includes a post-condition P.POST, a set of subtask $P.SUBT_1 \ldots P.SUBT_M$*. a set of input role types* $P.INPUT_1 \ldots P.INPUT_M$*, where P.#=M, and an output role type P.OUTPUT.*

Note that the input and output fields represent the sorts (types) of the corresponding parameters. Note also that the preconditions are not included in this definitions, this happens because an UPML precondition is a dynamic condition on inputs and thus it can not be checked statically to determine if a PSM can be used to solve a given task (it must be checket at run-time).

Now we can define the competence matching rule formally as follows.

DEFINITION 3: **COMPETENCE MATCHING RULE (~)**
A task T matches a PMS P (T~P) if the following conditions hold:

i **IF** P.POST = $C_1 \vee C_2 \vee \ldots \vee C_n$ **THEN** $\exists$i such that $C_i \rightarrow$ T.GOAL
ELSE *P.POST $\rightarrow$ T.GOAL*

ii **IF** P.# = T.# **and** $\forall$i $P.INPUT_i \supseteq T.INPUT_i$

iii P.OUTPUT $\subseteq$ T.OUTPUT

iv $\forall$i such that $P.SUBT_i$ exists a PMS P_j such that $P.SUBT_i \sim P_j$.

The first condition checks if T.GOAL can be deduced by P.POST. If the post-condition has the form $C_1 \vee C_2 \vee \ldots \vee C_n$ then each C_i expresses a possible result of the PSM computation and the matching succeed if $\exists$i such that $C_i \rightarrow$ G. Otherwise, the PSM is deterministic and we have only to check if *P.POST $\rightarrow$ T.GOAL.* Note that a non deterministic PSM matches a task also if $\exists$j such that $C_j \wedge \neg$T.GOAL i. This fact express the intuition that in general we cannot statically guarantee that a PSM (an algorithm) always terminates successfully. For instance we cannot prove that a search algorithm always succeeds (when it fails ¬T.GOAL can be true). Note that in our approach the post-condition deals with all the possible outputs of a PSM and thus also the failures are considered.

The second and the third conditions (ii and iii) state that the input parameters of the task must match the input parameters of the PSM and that this also holds for the output; we assume that the operator $\supseteq$ represents the notion of sub-typing of UPML.
Finally, the last condition (iv) encodes the recursive nature of the competence matching rule which must be executed for all the subtasks of the PSM which is considered.

In our experiment we have implemented this matching rule in LOOM. In the following we assume that the reader is familiar with the main aspects and concepts of description logic (Heinsohn et al. 1994) which constitute the basic components of LOOM (Brill 1993).

A PSM post-condition is translated into a LOOM concept which expresses the type of the result of the PSM. UPML sorts are translated into primitive LOOM concepts and input roles are defined as instances of these concepts according to their type. Goals are translated into LOOM concepts which model the structure of the results expected from tasks.

The matching rule is implemented as a set of operations on the database of concepts defining sorts, PSMs and tasks. Rules (ii and iii) are implemented as tests on the database of LOOM concepts. Rule (i) is implemented as a *subconcept-p* operation: given a concept that represents the goal of a task we search for a concept representing the post-condition of a PSM which subsumes it. If such concept exists it means that T.POST → G.GOAL. The matching procedure is recursively applied to all the subtasks.

Since concept subsumption is decidable in LOOM (this holds for at least a well defined subset of its description logic), the matching rule in DEFINITION 3 can also be made decidable. Of course to obtain this result the UPML competence specification must be translated into the subset of LOOM descriptions which guarantees a decidable subsumption algorithm. To obtain this result we need to restrict UPML which is based on first order logic, individuating a decidable subset. Although, this subset of UPML has still not been defined (it will be source of future research), the advantage of this approach is that we can apply results which have been achieved concerning concept subsumption in description logic (Heinsohn et al. 1994) (this issues has been investigated deeply in the literature) to our specific context of competence matching in UPML.

3.1. Competence Matching in LOOM.

We illustrate this approach considering the search task presented in the previous section. The search task has the following specification:

T_{search}.GOAL ::= solution_state (OUTPUT)
T_{search}.OUTPUT ::= State

where we assume that the predicate solution_state ranges on the sort *State* and both are defined in an associated search ontology. Note that the goal does not depend from the input. Then we consider the PSM search; the post-condition of the PSM search is:

P_{search}.POST ::= OUTPUT = $\emptyset \vee$ solution_state (OUTPUT)
P_{search}.OUTPUT ::= State

This post-condition expresses the fact that the search PSM can return a correct solution or can fail. Note that the post-condition imposes a restriction on the type of the output argument that in this case is State. The restriction says that the output must be a solution state.

This situation can be translated in LOOM as follows. UPML elementary sorts (*State* and *Input*) are translated into LOOM primitive concepts since the sort Input indicates a generic input it is not necessary to translate this explicitly in LOOM where we have a basic concept *Thing* indicating an unspecified type. Thus we only need a LOOM definition for State.

(defconcept State)

The unary predicates such as solution_state are represented as primitive concepts derived from state.

(defconcept Solution_State :is-primitive State)

The semantics of the *:is-primitive* operator guarantees that an instance of the defined concept, for instance *Solution_State* is also an instance of *State* but not vice-versa.

The post-condition is a disjunction between two conditions thus we need two separate concepts to define it.

(defconcept searchPSM-C1 :is EMPTY)
(defconcept searchPSM-C2 :is Solution_State)

where EMPTY is a conceps that only contains the NULL element and it is defined as follows:

(defconcept EMPTY :is (:one-of NULL))

If we consider the goal T_{search}.GOAL and the LOOM concept which represents this goal:

(defconcept searchGOAL :is Solution_State)

the following queries can be used to match this PSM:

(subconpect-p (fc searchPSM-C1) (fc searchGOAL))
(subconpect-p (fc searchPSM-C2) (fc searchGOA))

Where *(subconpect-p concept1 concept2)* performs the subsumption test between *concept1* and *concept2*, and *fc* retrieve a concept object given its name. These queries must be extended to deal with all the other candidate PSM and with the subtask of P_{search}.

4. Related Work and Conclusion

UPML is a specialised specification language for describing KBSs thus it is at a different level of abstraction with respect to general purpose specification languages such as Z. For instance it is possible to map the conceptual model of UPML onto the generic architecture and formalisation language of Z. For the same reason UPML is also a specialised language for describing software architectures. In fact PSMs are a special type of software architectures ([Shaw & Garlan, 1996]): software architectures for describing the reasoning part of KBSs.

We have presented an approach to select problem solving libraries based on semantic matching. The approach consists of extending a problem solving library with a competence specification expressed in description logic. In this scenario semantic matching is decidable and it is performed exploiting concept subsumption in description logic.

An issue that should be investigated more deeply is the relation of these competence matching techniques with the polymorphic type matching techniques exploited in object oriented programming languages. The main efforts concerning future work will be oriented to define a decidable language for specifying competence of PSM and tasks trying to integrate AI techniques and type matching operators in programming languages.

References

Benjamins, V. R., Plaza, E., Motta, E., Fensel, D., Studer, R., Wielinga, B., Schreiber, G., and Zdrahal, Z. (1998). An Intelligent Brokering Service for Knowledge-Component Reuse on the World-WideWeb. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW«98)*, Banff, Canada, April 18-23, 1998.

Breuker, J. and Van de Velde, W. (eds.): (1994). The CommonKADS Library for Expertise Modeling, IOS Press, Amsterdam, The Netherlands, 1994.

Brill, D. (1993). The Loom Reference Manual. Technical Report. University of southern California, 1993.

Chandrasekaran, B., Johnson, T.R., and Smith, J. W. (1992). Task Structure Analysis for Knowledge Modeling, Communications of the ACM, 35(9):124—137, 1992.

Fensel, D., Motta, E., Decker, S., and Zdrahal, Z. (1997). The use of Ontologies for Specifying Tasks and Problem Solving Methods: A Case Study. In R. Benjamins and E. Plaza (Editors). *Knowledge Acquisition, Modeling, and Management*. *Proceedings of the 10th European Workshop, EKAW '97.* Lecture Notes in Artificial Intelligence 1319, Springer-Verlag.

Fensel, D. and Motta, E. (1998). Structured Development of Problem Solving Methods. Available from http://ftp.aifb.uni-karlsruhe.de/pub/mike/dfe/spool/stepwise.{pdf, ps}.

Fensel,.D., Benjamins, V.R., Decker S., Gaspari, M., Groenboon, R., Grosso, W., Motta, E,. Plaza, E., Schreiber, G., Studer, R., Wielinga, B. (1999a). The Unified Problem-Solving Method Description Language UPML. IBROW-3 Esprit Project 27169. Project Deliverable 1.1 Chapter 1.

Fensel,.D., Benjamins, V.R., Decker S., Gaspari, M., Groenboon, R., Grosso, W., Motta, E,. Plaza, E., Schreiber, G., Studer, R., Wielinga, B. (1999b). The Component Model of UPML in a Nutshell. In Proceedings of the 1st Working IFIP Conference on Software Architectures (WICSA), San Antonio, Texas, USA, Febrary 1999.

Fensel, D., Benjamins, R., Motta, E., and Wielinga, B.: UPML: A Framework for knowledge system reuse. In Proceedings of the International Joint Conference on AI (IJCAI-99), Stockholm, Sweden, July 31 - August 5, 1999.

Fensel, D. and van Harmelen F.: (1994). A Comparison of Languages which Operationalize and Formalize KADS Models of Expertise, The Knowledge Engineering Review, 9(2), 1994.

Fensel, D. (1995) Formal Specification Languages in Knowledge and Software Engineering, The Knowledge Engineering Review, 10(4), 1995.

Gruber, T. R., Olsen, G. R., and Runkel, J. (1996). The configuration design ontologies and the VT elevator domain theory. *International Journal of Human-Computer Studies 44 (3/4)*.

Gruber T. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition, 5,* pp. 199-220.

Heinshon, J., Kudenko, D., Nebel, B., Profitlich, H., (1994). An Empirical Analysis of Terminological Representation Systems, Artificial Intelligence 68 (2) 1994 367-397.

Kifer, M., Lausen, G. and Wu, J.: (1995). Logical Foundations of Object-Oriented and Frame-Based Languages, Journal of the ACM, vol 42, 1995.

Marcus, S. (ed.): (1998) Automating Knowledge Acquisition for Experts Systems, Kluwer Academic Publisher, Boston, 1988.

Motta E. (1997). Reusable Components for Knowledge Models. *PhD Thesis*. Knowledge Media Institute. The Open University. UK. November 1997. Available from URL http://kmi.open.ac.uk/~enrico/thesis/thesis.html.

Motta, E., Fensel, D., Gaspari, M., Benjamin, R. (1999). Specification of Knowledge Components for Reuse. To appear in Proceedings of SEKE 1999.

Motta, E. and Zdrahal, Z.(1998). An approach to the organisation of a library of problem solving methods which integrates the search paradigm with task and method ontologies. To appear in the *Special Issue of the International Journal of Human-Computer Studies on Problem Solving Methods* (Benjamins and Fensel editors).

Musen, M.A. (1998). Modern Architectures for Intelligent Systems: Reusable Ontologies and Problem-Solving Methods. In C.G. Chute, Ed., 1998 AMIA Annual Symposium, Orlando, FL, 46-52. 1998.

Puppe, F. (1993). Systematic Introduction to Expert Systems: Knowledge Representation and Problem-Solving Methods, Springer-Verlag, Berlin, 1993.

Schreiber, A. TH., Wielinga, B. Akkermans, J.M., Van De Velde, W. and de Hoog, R: (1994). CommonKADS. A Comprehensive Methodology for KBS Development, IEEE Expert, 9(6):28—37, 1994.

Shaw M. and Garlan, D: (1996). Software Architectures. Perspectives on an Emerging Discipline, Prentice-Hall, 1996.